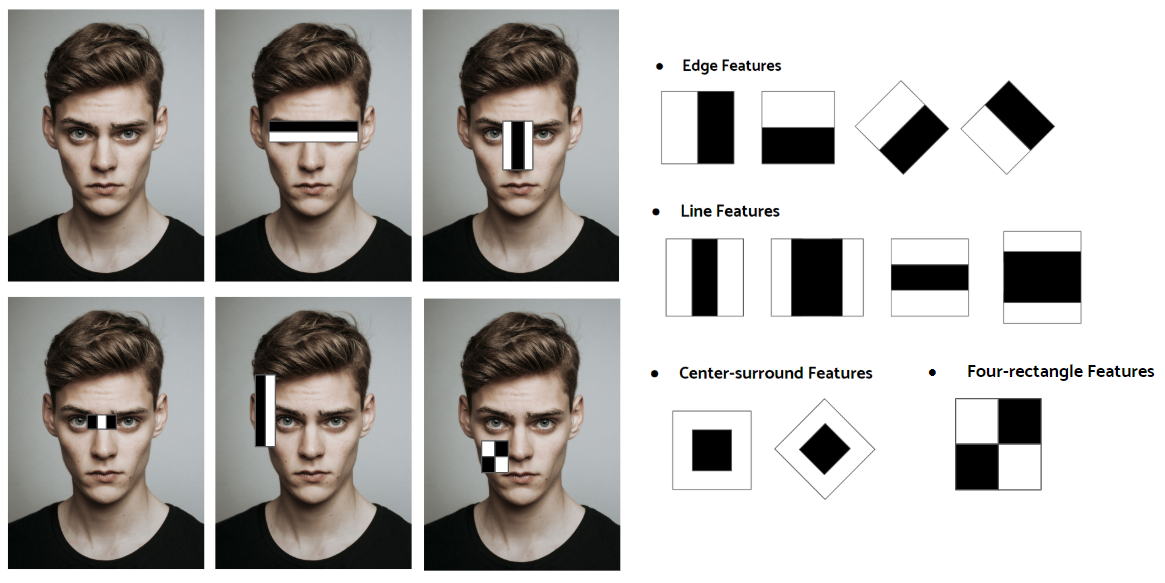
Лабораторная работа №9

# Обнаружение лиц

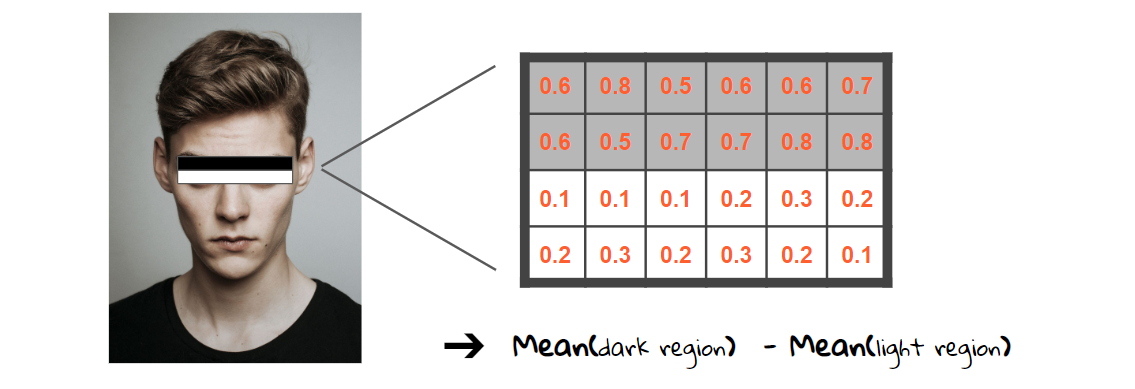
Последняя особенность, которую мы собираемся увидеть, - это лицо. Обнаружение лица - это технология, идентифицирующая присутствие и положение человеческих лиц на цифровых изображениях. Я хочу, чтобы вы отличали обнаружение лиц от распознавание лиц, что указывает на обнаружение личности человека по его или ее лицу. Таким образом, обнаружение лица не может сказать нам, кому принадлежит обнаруженное лицо.

## Каскадный классификатор Хаара

Обнаружение лица - это, в основном, задача классификации, поэтому она обучена определять, существует ли целевой объект или нет. А также **Каскадный классификатор Хаара** является одной из моделей обнаружения лиц, доступных в OpenCV. Это предварительно обученная модель, что означает, что она уже прошла обучение с тысячами изображений. 4 ключевых момента для понимания этого алгоритма Особенности извлечения Haar, целостное изображение, Adaboost, а также каскадные классификаторы.



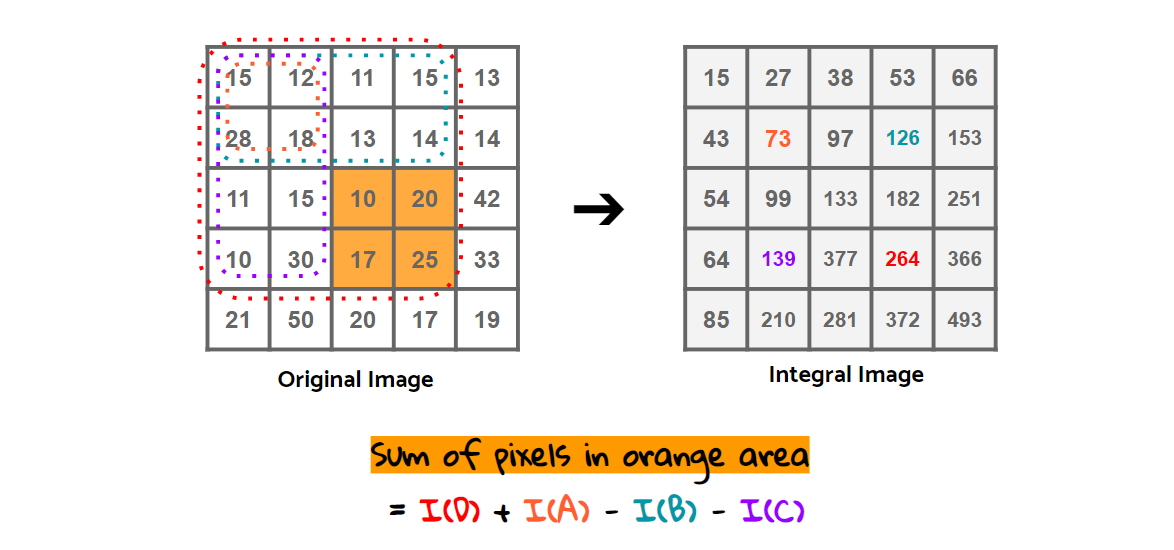
Хаар-подобные чертыэто фильтры изображений или ядра изображений, используемые при обнаружении объектов, и примеры приведены выше. Они обязаны своим именем их интуитивному сходству с вейвлетами Хаара, которые первоначально были предложеныАльфред Хаар, Во время обнаружения мы передаем окно на изображение и выполняем сверточную операцию с фильтрами, чтобы увидеть, есть ли функция, которую мы ищем, на изображении (<https://vimeo.com/12774628>) .



Так, как мы решаем, есть ли требуемая особенность или нет в данной области? Давайте посмотрим на картинку выше. У нас есть ядро, верхняя половина которого темная, а нижняя светлая. Затем мы получаем среднее значение пикселей для каждой области и вычитаем разрыв между ними. Если результат превышает пороговое значение, скажем, 0,5, то мы заключаем, что есть функция, которую мы обнаруживаем. Мы повторяем этот процесс для каждого ядра, перемещая окно по изображению.

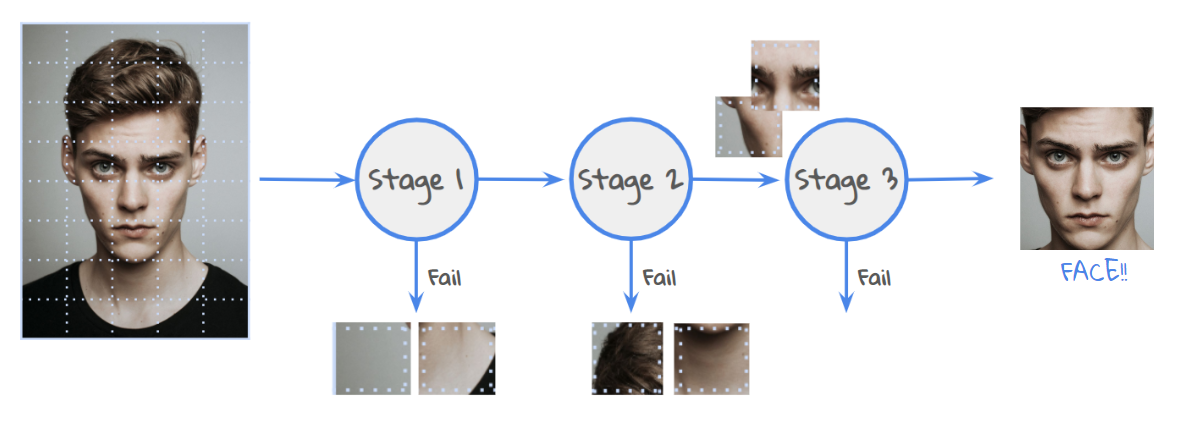
Хотя это не сложное вычисление, общее количество вычислений становится огромным, если рассматривать его в целом изображении. Если вы увидели видео, упомянутое выше, вы получите интуицию в подсчете суммы. И это где целостное изображение вступает в игру. Интегральное изображение - это способ представления изображения, который используется для ускорения и повышения эффективности оценки функции.

Как вы можете видеть ниже, слева находятся пиксели изображения, а справа - цельное изображение. Начиная с левой верхней точки, он вычисляет накопленную сумму пикселей в данной прямоугольной области. На интегральном изображении сумма пикселей в рамке с пунктирной линией написана в нижней правой точке прямоугольника справа.



С помощью этой предварительно рассчитанной таблицы мы можем просто получить суммированное значение для определенной области по значениям под прямоугольников (красный, оранжевый, синий и фиолетовый прямоугольник).

Таким образом, мы решили вычислительную стоимость с интегральным изображением. Но мы еще не закончили. Подумайте о том, когда окно обнаружения находится на пустом фоне, где нет объекта или лица. Это все равно будет пустой тратой времени, если он выполнит тот же процесс в такой части. И еще один удар делает этот детектор быстрее. Внедрение каскадный классификатор с Adaboost!



Каскадный классификатор строит ступенчатые ступени и дает порядок среди хаароподобных признаков. Основные формы функций реализуются на ранних этапах, а более сложные - только для перспективных регионов. И на каждом этапе модель Adaboost будет обучаться путем объединения слабых учеников. Если на предыдущем этапе подраздел или подокно классифицируется как «не похожая на лицо область», он отклоняется на следующем этапе. Таким образом, мы можем только рассмотреть выживших и достичь гораздо более высокой скорости.

Для этого мы применяем каждую функцию на всех обучающих изображениях. Для каждого признака он находит наилучший порог, который классифицирует грани на положительные и отрицательные. Очевидно, что будут допущены ошибки или неверные классификации. Мы выбираем объекты с минимальной частотой ошибок, что означает, что они являются объектами, которые наиболее точно классифицируют изображения лица и не лица. (Процесс не так прост, как это. Каждому изображению вначале придается равный вес. После каждой классификации вес неправильно классифицированных изображений увеличивается. Затем происходит тот же самый процесс. Вычисляются новые коэффициенты ошибок. А также новые веса. Этот процесс продолжается до тех пор, пока не будет достигнута требуемая точность или частота ошибок или не будет найдено необходимое количество признаков).

Окончательный классификатор представляет собой взвешенную сумму этих слабых классификаторов. Его называют слабым, потому что он сам по себе не может классифицировать изображение, но вместе с другими образует сильный классификатор. В документе говорится, что даже 200 функций обеспечивают обнаружение с точностью 95%. Их окончательной установки около 6000 объектов. (Представьте себе сокращение с 160000+ функций до 6000 функций. Это большой выигрыш).

Итак, теперь вы берете изображение. Возьмите каждое окно 24x24. Примените к нему 6000 функций. Проверьте, лицо это или нет. Вау. Не слишком ли это неэффективно и отнимает много времени? Да, это так. У авторов есть хорошее решение для этого.

В изображении большая часть изображения не является областью лица. Поэтому лучше иметь простой метод проверки, не является ли окно областью лица. Если это не так, выбросьте его в один выстрел и не обрабатывайте его снова. Вместо этого сосредоточьтесь на тех областях, где может быть лицо. Таким образом, мы тратим больше времени на проверку возможных областей лица.

Для этого они ввели понятие каскада классификаторов. Вместо того чтобы применять все 6000 объектов в окне, объекты группируются в различные этапы классификаторов и применяются один за другим. (Обычно первые несколько этапов содержат гораздо меньше функций). Если окно не проходит первый этап, откажитесь от него. Мы не рассматриваем остальные особенности на нем. Если он проходит, примените второй этап функций и продолжите процесс. Окно, которое проходит все стадии, является областью лица. Как же это план такой!

Детектор авторов имел 6000+ признаков с 38 стадиями с 1, 10, 25, 25 и 50 признаками в первых пяти стадиях. (Две функции на приведенном выше изображении фактически получены как лучшие две функции из Adaboost). По данным авторов, в среднем 10 функций из 6000+ оцениваются в каждом подокне.

Таким образом, это простое интуитивное объяснение того, как работает распознавание лиц Виолы-Джонса. Прочтите статью для получения более подробной информации или ознакомьтесь со ссылками в разделе дополнительные ресурсы.

## Haar-cascade Detection in OpenCV

OpenCV предоставляет метод обучения (см. Cascade Classifier Training) или предварительно обученные модели, которые можно прочитать с помощью метода cv::CascadeClassifier::load. Предварительно обученные модели находятся в папке data в установке OpenCV или могут быть найдены здесь.

В следующем примере кода будут использоваться предварительно обученные каскадные модели Хаара для обнаружения лиц и глаз на изображении. Во-первых, резюме::создан CascadeClassifier и необходимые загружен XML-файл использование CV::CascadeClassifier::способ загрузки. После этого обнаружение выполняется с помощью метода cv::CascadeClassifier::detectMultiScale, который возвращает граничные прямоугольники для обнаруженных лиц или глаз.

Этот обучающий код показан строками ниже.

**from \_\_future\_\_ import print\_function**

**import cv2 as cv**

**import argparse**

**def detectAndDisplay(frame):**

**frame\_gray = cv.cvtColor(frame, cv.COLOR\_BGR2GRAY)**

**frame\_gray = cv.equalizeHist(frame\_gray)**

**#-- Detect faces**

**faces = face\_cascade.detectMultiScale(frame\_gray)**

**for (x,y,w,h) in faces:**

**center = (x + w//2, y + h//2)**

**frame = cv.ellipse(frame, center, (w//2, h//2), 0, 0, 360, (255, 0, 255), 4)**

**faceROI = frame\_gray[y:y+h,x:x+w]**

**#-- In each face, detect eyes**

**eyes = eyes\_cascade.detectMultiScale(faceROI)**

**for (x2,y2,w2,h2) in eyes:**

**eye\_center = (x + x2 + w2//2, y + y2 + h2//2)**

**radius = int(round((w2 + h2)\*0.25))**

**frame = cv.circle(frame, eye\_center, radius, (255, 0, 0 ), 4)**

**cv.imshow('Capture - Face detection', frame)**

**parser = argparse.ArgumentParser(description='Code for Cascade Classifier tutorial.')**

**parser.add\_argument('--face\_cascade', help='Path to face cascade.', default='haarcascades/haarcascade\_frontalface\_alt.xml')**

**parser.add\_argument('--eyes\_cascade', help='Path to eyes cascade.', default='haarcascades/haarcascade\_eye\_tree\_eyeglasses.xml')**

**parser.add\_argument('--camera', help='Camera divide number.', type=int, default=0)**

**args = parser.parse\_args()**

**face\_cascade\_name = args.face\_cascade**

**eyes\_cascade\_name = args.eyes\_cascade**

**face\_cascade = cv.CascadeClassifier()**

**eyes\_cascade = cv.CascadeClassifier()**

**#-- 1. Load the cascades**

**if not face\_cascade.load(cv.samples.findFile(face\_cascade\_name)):**

**print('--(!)Error loading face cascade')**

**exit(0)**

**if not eyes\_cascade.load(cv.samples.findFile(eyes\_cascade\_name)):**

**print('--(!)Error loading eyes cascade')**

**exit(0)**

**camera\_device = args.camera**

**#-- 2. Read the video stream**

**cap = cv.VideoCapture(camera\_device)**

**if not cap.isOpened:**

**print('--(!)Error opening video capture')**

**exit(0)**

**while True:**

**ret, frame = cap.read()**

**if frame is None:**

**print('--(!) No captured frame -- Break!')**

**break**

**detectAndDisplay(frame)**

**if cv.waitKey(10) == 27:**

**break**

Файлы классификатора Haar Cascade в основном поставляются вместе с OpenCV. Вероятно, вы можете найти их в папке OpenCV на вашем компьютере, или тут:

<https://github.com/jjone36/vision_4_beginners/tree/master/haarcascades>

(немного дополнительной литуратуры, рекомендуемой Руководством:

1. Paul Viola and Michael J. Jones. Robust real-time face detection. International Journal of Computer Vision, 57(2):137–154, 2004. [[218]](https://docs.opencv.org/3.4/d0/de3/citelist.html#CITEREF_Viola04)
2. Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In Image Processing. 2002. Proceedings. 2002 International Conference on, volume 1, pages I–900. IEEE, 2002. [[127]](https://docs.opencv.org/3.4/d0/de3/citelist.html#CITEREF_Lienhart02)
3. Video Lecture on [Face Detection and Tracking](https://www.youtube.com/watch?v=WfdYYNamHZ8)
4. An interesting interview regarding Face Detection by [Adam Harvey](https://web.archive.org/web/20171204220159/http:/www.makematics.com/research/viola-jones/)
5. [OpenCV Face Detection: Visualized](https://vimeo.com/12774628) on Vimeo by Adam Harvey

)

Далее мы собираемся создать функцию, которая распознает лицо и рисует вокруг него прямоугольник. Чтобы определить лицо, мы можем использовать метод detectMulitiScale() классификатора face\_cascade что мы загрузили выше. Он возвращает четыре точки указанной области, поэтому мы нарисуем прямоугольник в этой позиции .scaleFactor параметр для уменьшения размера изображения при каждом масштабе изображения и minNeighbors на сколько соседей должен быть обучен каждый кандидатный прямоугольник. Теперь давайте применим эту функцию к изображению и увидим результат.

**import cv2**

**import numpy as np**

**# Load Cascade filter**

**face\_cascade = cv2.CascadeClassifier('haarcascades/haarcascade\_frontalface\_default.xml')**

**# Create the face detecting function**

**def detect\_face(img):**

**img\_2 = img.copy()**

**face\_rects = face\_cascade.detectMultiScale(img.copy(),**

**scaleFactor = 1.1,**

**minNeighbors = 4)**

**for (x, y, w, h) in face\_rects:**

**cv2.rectangle(img\_2, (x, y), (x+w, y+h), (255, 255, 255), 3)**

**return img\_2# Detect the face**

**# Load the image file and convert the color mode**

**avengers = cv2.imread('maxresdefault.jpg')**

**avengers = cv2.cvtColor(avengers, cv2.COLOR\_BGR2GRAY)# Detect the face and plot the result**

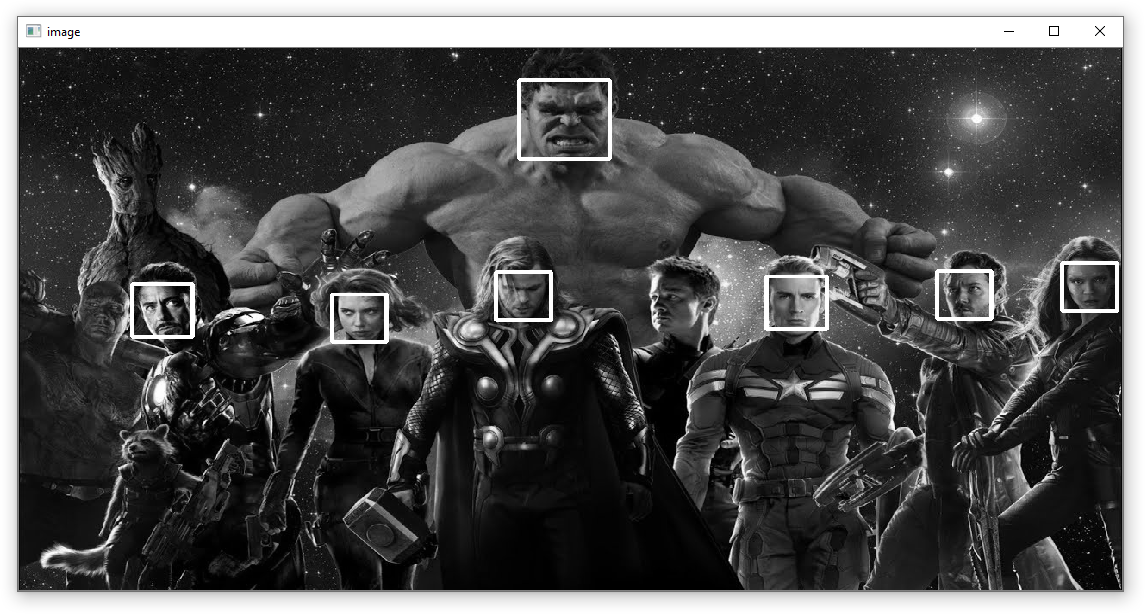
**detected\_avengers = detect\_face(avengers)**

**cv2.namedWindow('image', cv2.WINDOW\_NORMAL)**

**cv2.imshow('image',detected\_avengers)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



Вопросы к лабораторной работе

1. привести подробное описание основных функций, рассмотренных в работе - функция, что делает, аргументы, параметры, возвращаемые результаты, типы данных и тп.
2. выполнить примеры, описанные в лабораторной - предоставить комментированный код, поэтапные результаты выполнения действий (исходные фото-, видео- данные для примеров берутся собственные, или из открытых источников или из базового набора библиотеки OpenCV)